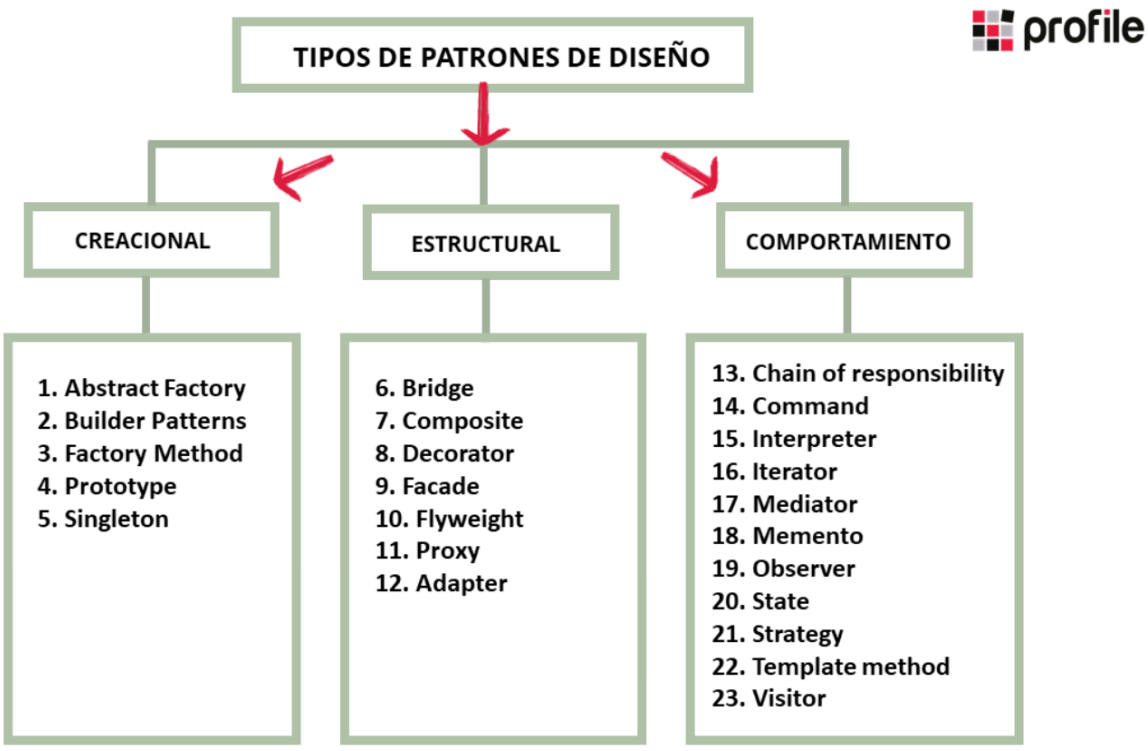
Patrones de diseño de software

Un patrón es la solución a un problema en un contexto en particular. Nos permite entender como adaptarlo al problema específico al cual se quiere aplicar. Los patrones facilitan la reutilización de diseños y arquitecturas software que han tenido éxito.
Christopher Alexander (1977): Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y describe la esencia de la solución a ese problema, de tal modo que pueda utilizarse esta solución un millón de veces más, sin siquiera hacerlo de la misma manera dos veces.
Existen muchos patrones de diseño de software detallados a continuación.



Patrones de diseño creacional

Los patrones de creación proporcionan diversos mecanismos de creación de objetos, que aumentan la flexibilidad y la reutilización del código existente.

Abstract Factory

En este patrón, una interfaz crea conjuntos o familias de objetos relacionados sin especificar el nombre de la clase.

Builder Patterns

Permite producir diferentes tipos y representaciones de un objeto utilizando el mismo código de construcción. Se utiliza para la creación etapa por etapa de un objeto complejo combinando objetos simples. La creación final de objetos depende de las etapas del proceso creativo, pero es independiente de otros objetos.

Factory Method

Proporciona una interfaz para crear objetos en una superclase, pero permite que las subclases alteren el tipo de objetos que se crearán. Proporciona instanciación de objetos implícita a través de interfaces comunes

Prototype

Permite copiar objetos existentes sin hacer que su código dependa de sus clases. Se utiliza para restringir las operaciones de memoria / base de datos manteniendo la modificación al mínimo utilizando copias de objetos.

Singleton

Este patrón de diseño restringe la creación de instancias de una clase a un único objeto.

Patrones estructurales

Facilitan soluciones y estándares eficientes con respecto a las composiciones de clase y las estructuras de objetos.

Adapter

Se utiliza para vincular dos interfaces que no son compatibles y utilizan sus funcionalidades. El adaptador permite que las clases trabajen juntas de otra manera que no podrían al ser interfaces incompatibles.

Bridge

En este patrón hay una alteración estructural en las clases principales y de implementador de interfaz sin tener ningún efecto entre ellas. Estas dos clases pueden desarrollarse de manera independiente y solo se conectan utilizando una interfaz como puente.

Composite

Se usa para agrupar objetos como un solo objeto. Permite componer objetos en estructuras de árbol y luego trabajar con estas estructuras como si fueran objetos individuales.

Decorator
Este patrón restringe la alteración de la estructura del objeto mientras se le agrega una nueva funcionalidad. La clase inicial permanece inalterada mientras que una clase decorator proporciona capacidades adicionales.

Facade
Proporciona una interfaz simplificada para una biblioteca, un marco o cualquier otro conjunto complejo de clases.

Flyweight
El patrón Flyweight se usa para reducir el uso de memoria y mejorar el rendimiento al reducir la creación de objetos. El patrón busca objetos similares que ya existen para reutilizarlo en lugar de crear otros nuevos que sean similares.

Proxy
Se utiliza para crear objetos que pueden representar funciones de otras clases u objetos y la interfaz se utiliza para acceder a estas funcionalidades

Patrones de comportamiento
El patrón de comportamiento se ocupa de la comunicación entre objetos de clase. Se utilizan para detectar la presencia de patrones de comunicación ya presentes y pueden manipular estos patrones.

Chain of responsibility
El patrón de diseño Chain of Responsibility es un patrón de comportamiento que evita acoplar el emisor de una petición a su receptor dando a más de un objeto la posibilidad de responder a una petición.

Command
Convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. Esta transformación permite parametrizar métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y respaldar operaciones que no se pueden deshacer.

Interpreter
Se utiliza para evaluar el lenguaje o la expresión al crear una interfaz que indique el contexto para la interpretación.

Iterator
Su utilidad es proporcionar acceso secuencial a un número de elementos presentes dentro de un objeto de colección sin realizar ningún intercambio de información relevante.

Mediator
Este patrón proporciona una comunicación fácil a través de su clase que permite la comunicación para varias clases.

Memento
El patrón Memento permite recorrer elementos de una colección sin exponer su representación subyacente.

Observer
Permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que está siendo observado.

State
En el patrón state, el comportamiento de una clase varía con su estado y, por lo tanto, está representado por el objeto de contexto.

Strategy
Permite definir una familia de algoritmos, poner cada uno de ellos en una clase separada y hacer que sus objetos sean intercambiables.

Template method
Se usa con componentes que tienen similitud donde se puede implementar una plantilla del código para probar ambos componentes. El código se puede cambiar con pequeñas modificaciones.

Visitor
El propósito de un patrón Visitor es definir una nueva operación sin introducir las modificaciones a una estructura de objeto existente.

Referencias
https://www.fdi.ucm.es/profesor/jpavon/poo/2.14pdoo.pdf
https://profile.es/blog/patrones-de-diseno-de-software/ - Abstract_Factory